On the design of Java based backend platforms for low latency IMS applications

Bruno Van Den Bossche, Filip De Turck, Bart Dhoedt and Piet Demeester
Ghent University - IBBT - IMEC, Department of Information Technology
Gaston Crommenlaan 8 bus 201, 9050 Gent, Belgium
Tel: +3293314900, Fax: +3293314899
Email: Bruno.VanDenBossche@intec.UGent.be

Abstract— **The Java programming language and more specifically the J2EE platform has evolved toward one of the important software frameworks for designing and toward implementing business logic on a telecom backend platform. However, Java suffers from a latency problem due to the automatic memory management and garbage collection. In this paper we show that these problems can be solved by tuning of the virtual machine.**

An important current Java technology is SIP Servlet, which allows fast development of applications using the Session Initiation Protocol (SIP). The SIP Servlet Specification, based on the well known HTTP Servlet concept, is part of the JAIN Initiative which resulted in a number of Java technologies optimized for the development of Telecom applications and IP Multimedia Subsystems (IMS). This paper evaluates both the functionality and performance of the SIP Servlet specification and implementations, focusing on low latency requirements.

I. Introduction

The Java language is highly structured, strongly typed and object oriented. It is not compiled to machine specific instructions but a *byte code*, which is then executed on a virtual machine, available for all sorts of platforms. Although this results in a certain performance penalty, it does make Java highly portable. Another feature with important implications is the use of a garbage collector. Java includes automatic memory management (garbage collection) as a part of the Java runtime. This means that many very common errors made by developers related to memory management cannot occur. Since the garbage collection is a part of the Java runtime it is not completely under the control of the application developer and it complicates the predictability of the garbage collection, which could result in pauses during the application execution.

Telecom applications often have very strict requirements regarding the throughput (e.g. the number of Voice over IP - VoIP - call setups a softswitch can process per second) and the latency (e.g. the setup of a call should be very fast). These requirements might indicate that Java would not be a good choice for the job as the behavior of the garbage collection and the virtual machine is not under control of the application developer and might conflict with the strict requirements.

To make Java more attractive to the telecom industry the JAIN [1], [2] (Java APIs for Integrated Networks) initiative is providing us with an extensive set of standardized APIs to facilitate the development and deployment of telecom services. One of the resulting technologies of the JAIN initiative is SIP Servlet which focuses on fast development of SIP based applications. Although JAIN does offer open standardized specifications tailored for telecom applications, it does not directly address the issues of the Java garbage collection. Fortunately most virtual machines allow to *tune* the behavior of the garbage collection, thus giving some control over the garbage collection back to the application deployers. This paper proposes tuning solutions optimized for the low latency requirements of typical telecom applications.

The structure of this paper is as follows: first IMS and SIP will be briefly explained in section II and a discussion of the SIP Servlet technology will be presented, highlighting the architecture and features in section III and relations with other existing technologies in section IV. Next a brief overview of the relevant Java Virtual Machine internals will be given in section V, followed by the obtained tuning options. The test setup and the results of the performance evaluation will be presented in sections VI and VII.

II. IP Multimedia Subsystems

A. Definition

The IP Multimedia Subsystem (IMS) is an open standardized multi-media architecture for mobile and fixed IP services [3]. It is a VoIP implementation based on a variant of SIP, and runs over IP. It is used by telecom operators to offer network controlled multimedia services. The aim of IMS is not only to provide new services but to provide all the services, current and future, that the Internet provides. A multi-media session between two IMS users, between an IMS user and a user on the Internet, and between two users on the Internet is established using exactly the same protocol. Moreover, the interfaces for service developers are also based on IP protocols such as the Session Initiation Protocol.

B. Session Initiation Protocol (SIP)

The Session Initiation Protocol (SIP) is defined in RFC3261 [4] and describes an application-layer control (signaling) protocol for creating, modifying, and terminating sessions with one or more participants. These sessions include Internet telephone calls, multimedia distribution, and multimedia conferences. RFC3261 also defines the use of SIP proxies and how they should interact with other SIP applications and proxies. We chose the SIP proxy as the use case for

the benchmarks discussed in this paper as it is an important participant in typical SIP Sessions.

The scenario used for the testing and benchmarks is the Proxy 200 test shown in Figure 1 defined in the SIPstone benchmark [5]. We are interested in the time it takes to set up a call, this is the time it takes from the initial INVITE of Alice until she receives an OK from Bob. After this Alice will send

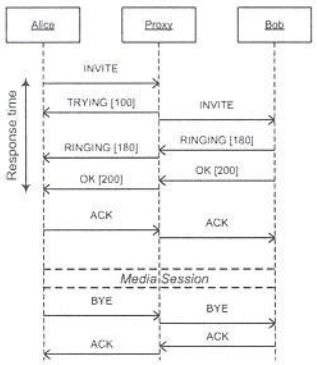


Fig. 1. Proxy 200 test

an acknowledgment to Bob saying she received the OK and the media session (e.g. voice or video conference) can start. When benchmarking no media session was initiated and the call was terminated immediately.

III. SIP Servlet

The SIP Servlet specification [6] provides us with a container in which Servlet based applications can be deployed. The specification was primarily defined to simplify the development of SIP enabled applications.

A. Architecture

Figure 2 shows the general architecture of a SIP Servlet Application Server. The server consists of a Servlet container providing the SIP Servlet APIs. In the container multiple applications can be deployed, each containing one or more SIP Servlets. Each application also contains a deployment descriptor which contains information about the application and tells the container which SIP messages it should direct to which servlets. The servlets can then process the SIP messages and if necessary pass them on to other servlets. Multiple servlets may process the same SIP message.

B. Relation to HTTP Servlet

The main difference between SIP Servlets and HTTP Servlets is the synchronicity of the calls. Where a HTTP Servlet only handles synchronous calls, a SIP Servlet also needs to handle asynchronous communications. Synchronous communication means that in a call the caller waits until the callee replies. In an asynchronous situation the caller does not always need to wait or the callee may initiate the communication himself when it is ready. A second important difference is that HTTP Servlets are only hosted on *origin servers* which generate a final response. SIP Servlets also

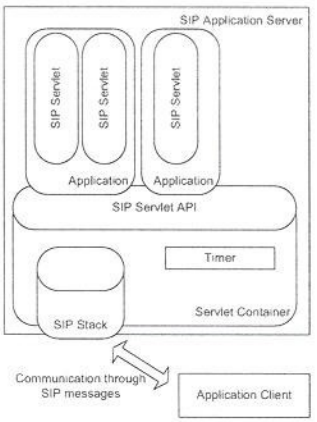


Fig. 2. SIP Servlet architecture

need to be able to route and initiate requests. Another key difference is that SIP Servlets sometimes need to generate multiple responses to one request (e.g. a provisional response followed by a final response) and they may receive both responses as well as requests.

IV. Related Technologies

SIP Servlet is not the only available Java technology that can be used for implementing the SIP Proxy application used in the benchmarks. JAIN SIP [7] specifies a SIP protocol stack in Java compliant to RFC 3261. This means an API is defined to handle and simplify any SIP related communications by supplying the necessary classes and methods.

JAIN SLEE [8] is a protocol agnostic Service Logic Execution Environment. Using the same basic concepts as J2EE and focusing on high throughput and low latency it tries to meet the strict requirements of telecom applications. This technology, in contrast to SIP Servlet, is not specifically designed for use as a SIP Application Server. It is possible however to enable the SLEE to be used in a SIP context by plugging in a SIP Resource Adapter. For a functional comparison of JAIN SLEE with J2EE and a performance evaluation we refer to previous work reported upon in [9].

V. Java Virtual Machine Internals

The used hardware platform always has a great influence on any performance evaluation, but so has the Java Virtual Machine. Of great importance is the garbage collection and memory management. Garbage collection may lead to the whole virtual machine being blocked. These pauses may not last too long or they cause a timeout when setting up a SIP call.

An object in the Virtual Machine is considered garbage if it can no longer be reached through any reference in the running program. So a simple garbage collection algorithm might traverse all objects and check for references. If an object has not got any references to it, it can be considered garbage. These objects can then be destroyed and the memory can be released to the virtual machine. However, this approach is not

very scalable and becomes very slow really fast when using larger heap sizes. More optimal collectors are available in

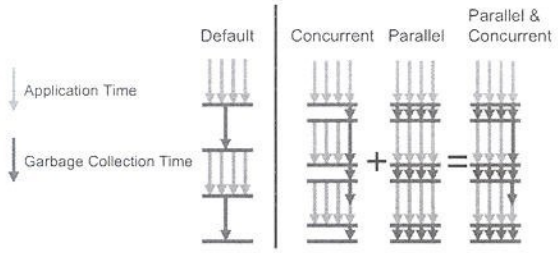


Fig. 3. The available garbage collectors

the current JVMs and allow for garbage collection to occur (partially) parallel with the execution of the application or spawn multiple threads which allows them to benefit from multi-cpu servers. Both type of collectors can also be run cooperatively as shown in figure 3 to even further improve the garbage collection performance. However, the more advanced the used garbage collector is, the more resources are required for the actual garbage collection. A trade-off has to be made between the resources spent on garbage collection and the possible benefits. The options specified in table I are specific

-Xmx512m	(1)
-XX:NewSize=32m	(2)
-XX:MaxNewSize=32m	(3)
-XX:MaxTenuringThreshold=0	(4)
-XX:SurvivorRatio=128	(5)
-XX:+UseParNewGC	(6)
-XX:+UseConcMarkSweepGC	(7)
-XX:+CMSIncrementalMode	(8)
-XX:+CMSIncrementalPacing	(9)
-XX:CMSIncrementalDutyCycleMin=0	(10)
-XX:CMSIncrementalDutyCycle=10	(11)

TABLE I

VIRTUAL MACHINE TUNING OPTIONS FOR LOW LATENCY BEHAVIOR.

to the Sun JVM, but other virtual machines offer similar tuning options which can be used to achieve the same effect.

The memory managed by the virtual machine is divided into multiple generations (Young, Tenured and Perm), depending on the age of the objects. This allows the garbage collector to take advantage of the fact that the vast majority of objects has a very short lifetime as it does not need to collect the older generations as often. Long living objects are moved into the next generation after a certain amount of time or a number of garbage collections. By specifying the sizes of the generations (1-3) and limiting the amount of time before an object is promoted to the next generation (4-5) we can achieve that objects lasting the whole call are moved to an older generation very fast. This is beneficial as the older generations are not garbage collected as much and thus the objects it contains are less often inspected. The garbage collector itself can also be tuned (6-11) to use multiple threads on multi-cpu machines and to work concurrently with the application execution for as long as possible. This allows to limit the time the execution of the virtual machine needs to be paused completely for garbage collection.

VI. TEST SETUP

Before we discuss the obtained results we will first give an overview of the test setup used.

A. Software Setup

For benchmarking purposes SIPp [10] is used. SIPp is a free Open Source test tool/traffic generator for the SIP protocol. It allows generating SIP traffic and to establish and release multiple calls. It can also read custom XML scenario files, describing from very simple to complex call flows. It features the dynamic display of statistics about running tests such as call rate, round trip delay, and message statistics, periodic CSV statistics dumps, TCP and UDP over multiple sockets or multiplexed with retransmission management and dynamically adjustable call rates. The cpu load was measured using the mpstat tool included in most Linux distributions. The SIP Servlet application Server used for the evaluation of the optimized garbage collection tuning was the BEA Weblogic SIP Server [11].

B. Hardware Setup

All tests were performed using a dual Opteron 242 HP DL 145 with 2GB of memory for the proxy. The clients were run on AMD athlonXP 1600+ machines with everything interconnected in a 100Mb switched ethernet network. All platforms were running Debian GNU/Linux with a 2.6 kernel and the Sun JDK 1.4.2 was used.

VII. SIP SERVLET PERFORMANCE RESULTS

This section gives an overview of the obtained test results. The SIP Servlet container was submitted to a number of test runs that allowed us to evaluate the garbage collection tuning options. When setting up a SIP call using the scenario specified in figure 1, the acknowledgment of the INVITE message should arrive within 50ms after sending it. Requirements specified by the telecom industry state that this should be true for 95% of the calls and that for 50% of the calls it should arrive within 25ms.

Figure 4 shows the performance results of the SIP Servlet application server when no special tuning parameters were set. It clearly shows that the 95th percentile rises very fast starting at 50 calls per second (caps). At this point multiple SIP messages start arriving concurrently which means some messages are first queued before they can be processed. A second problem with this result is the low cpu usage. At the higher call rates the cpu-usage stays relatively low but some calls do time out which is of course not desirable. When taking into account that 95% of the calls should be answered within 50ms this configuration would only be able to handle 130 caps.

Figure 5 shows the results of a test run with the optimized low latency options from Table I. The first difference to notice is the much slower increase of the 95th percentile. Although there still is an increased delay starting at 70 caps the increase

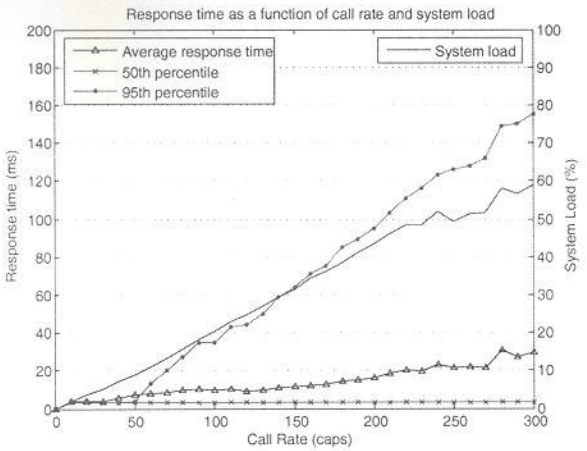


Fig. 4. SIP Servlet results on a dual cpu machine with default garbage collection.

is much less steep. As the average garbage collection pauses are much lower than with the default options, the majority of the calls will not be delayed too long. With the default garbage collection options pauses up to 1 second could occur as opposed to less than 10ms for the optimized options. Secondly

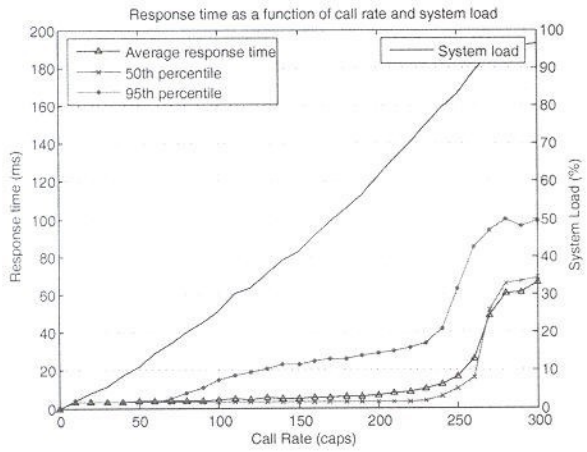


Fig. 5. SIP Servlet results on a dual cpu machine with optimized garbage collection tuning as specified in Table I and thread local allocation enabled.

we notice the cpu usage is higher and reaches almost 100% at 250 caps. Starting at this callrate some calls do timeout. Fortunately we do get the better response times in return. Obviously the more advanced garbage collector algorithms require more system resources, but it allows them to do a much better job. The requirement for the 95th percentile is now met up to 240 caps, which is a significant increase compared to the 130 caps reached with the default garbage collector. Although highly optimized C or C++ implementations of a SIP Proxy can achieve higher call rates they do not offer the same flexibility in application design.

As of Java 5.0 a number of new high level garbage collection tuning options have been introduced. The goal of the newly introduced options is to simplify the tuning by minimizing the number of required options to pass on the command line when starting the Java virtual machine. Currently, test results show the new options do improve the default garbage collection behavior but are not (yet) capable to achieve equivalent results of the tuning options presented in this paper.

VIII. Conclusions and Future Work

Java technologies such as SIP Servlet are currently available for telecom and IMS platform design. In this paper, it is shown that performance problems caused by the garbage collector of the Java Virtual Machine can be solved by specifying tuning options that will improve the low latency behavior of the garbage collector.

As telecom applications have very strict requirements when it comes to low latency and high throughput a multimedia call setup was chosen as a real world test case for the validation of the proposed optimizations and the SIP Servlet technology was selected due to its interesting features for IMS backend platform design. The evaluation results show that Java and SIP Servlet can meet the strict requirements of telecom applications if appropriate tuning for the Java Virtual Machine is used.

Acknowledgment

We would like to thank BEA for providing us with a license for their Communications platform and the stimulating discussions. The research presented in this paper is partially funded by the IBBT T-Case project. F. De Turck is a postdoctoral Fellow of the Fund for Scientific Research - Flanders (F.W.O.-Vlaanderen).

References

[1] Sun Microsystems, "Java API's for Integrated Networks," http://java.sun.com/products/jain/.
[2] J. de Keijzer, D. Tait, and R. Goedman, "JAIN: A New Approach to Services in Communication Networks," *IEEE Communications Magazine*, vol. 38, no. 1, pp. 94–99, January 2000.
[3] 3GPP, "3GPP A Global Initiative," http://www.3gpp.org/.
[4] J. Rosenberg, H. Schulzrinne, G. Camarillo, A. Johnston, J. Peterson, R. Sparks, M. Handley, and E. Schooler, "Session Initiation Protocol," 2002, http://www.ietf.org/rfc/rfc3261.txt?number=3261.
[5] H. Schulzrinne, S. Narayanan, J. Lennox, and M. Doyle, "SIPstone - Benchmarking SIP Server Performance," April 2002, http://www.sipstone.org/.
[6] Sun Microsystems, "SIP Servlet API," http://jcp.org/en/jsr/detail?id=116.
[7] ——, "JAIN SIP Specification," http://www.jcp.org/en/jsr/detail?id=32.
[8] ——, "JAIN SLEE API Specification," http://jcp.org/en/jsr/detail?id=22.
[9] B. Van Den Bossche, F. De Turck, B. Dhoedt, T. Pollet, B. Van Vlerken, J. Moreels, N. Janssens, P. Demeester, and D. Colle, "Evaluation of current java technologies for telecom backend platform design," in *Proceedings of the 2005 International Symposium on Performance Evaluation of Computer and Telecommunication Systems*, July 2005, pp. 699–709.
[10] Hewlett-Packard, "SIPp : SIP benchmarking utility," http://sipp.sourceforge.net/.
[11] BEA Systems, "BEA WebLogic SIP Server: Converged IP multimedia communications services," http://www.bea.com/framework.jsp?CNT=index.htm&FP=/content/products/wlcom/sip/.

ICT 2006

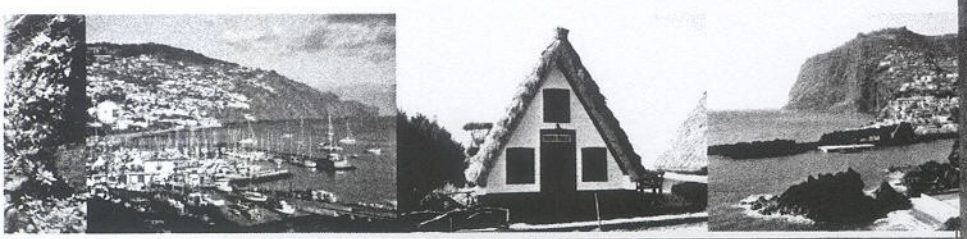
13th International Conference on Telecommunications

9-12 May, Funchal
Madeira Island
Portugal

Organized by:

instituto de telecomunicações

UNIVERSIDADE da MADEIRA




ICT
2006
13th International Conference
on Telecommunications
9-12 May, Funchal
Madeira Island
Portugal
it
instituto de
telecomunicações